SYSTEM AND METHOD OF GENERATING
COMPUTER SYSTEM PERFORMANCE REPORTS

TECHNICAL FIELD OF THE INVENTION

5   This invention relates generally to computer system

performance studies, and more particularly to the monitoring

and reporting of operation characteristics of components of a

computer system.

10   ### BACKGROUND OF THE INVENTION

        A modern computer system is typically a complicated

combination of software and hardware that has many different

components for performing various functions and supporting

various features. The optimal performance of a computer

15   system often can be obtained only by continuously monitoring

the health and performance of the components of the computer

system, and correcting problems identified through such

monitoring.

        The need for continuous performance studies is present

20   not only in operating an existing computer system but also in

developing computer software and hardware products. For

instance, during the development of an operating system, such

as the Windows NT operating system by Microsoft Corporation,

various components of the operating system are constantly

25   being tested by subjecting them to strenuous operating

conditions and observing whether they can withstand the heavy

usage without failure. Such a performance study, often termed

"stress testing," helps the software developers to identify

the weak spots or defects in the components of the operating

system and provides valuable information as to the causes of
failure.

In this regard, the collection of meaningful data
regarding the operation characteristics of the system

5   components and the compilation of the collected data into
reports in useful formats are critical aspects of an effective
system performance study. For each of the components being
monitored, there may be a number of statistical variables that
are of interest and should be tracked. The collected

10   statistical data then have to be presented in easy-to-
understand formats to facilitate identification of the status
of the components and diagnosis of problems. Moreover, the
results of a performance study often are to be reviewed by
different levels of management. To that end, it is often

15   necessary to provide reports that summarize the results of the
performance study on different levels of abstraction to suit
the different information needs of the management. For
instance, in a network environment, a network administrator
may want to know the total number of calls processed by a

20   given server in the network, while a top-level manager may
only be interested knowing the general health of the network.

Existing reporting tools for reporting the results of
system performance studies, however, do not satisfactorily
meet these reporting needs. For instance, in the example of

25   software development of the operating system, the development
team is divided into groups, with each group responsible for
one or more components of the operating system. Stress tests

for various components are run on a plurality of computers, and the states of the stressed components are closely monitored by the responsible groups. Generally, each group tracks and reports a variety of statistics collected from

5 machines running the stress tests for its components, and shares the information within itself and with other groups. To that end, each group typically implements its own ad hoc tracking and reporting applications. Due to the various types of statistical data tracked by different groups and the

10 inconsistent ways the data are reported, the stress data provided by one group often cannot be readily used with data provided by other groups for analysis and summary purposes. Moreover, information on critical system attributes necessary for monitoring system performance and health is often not

15 uniformly tracked and in some cases are simply omitted from the tracking tools of some individual groups. Such an inconsistency in tracking critical system attributes makes it difficult to establish a baseline for system evaluation.

The need to generate useful reports from the collected

20 data poses another problem. In many cases, there are formatting requirements for stress reports that have to be adhered to. Experience has shown that constructing stress reports that meet the given formatting requirements is a very time-consuming task that is prone to errors. Often times such

25 reports are generated by hand, requiring sometimes hours to assemble and format the relevant data. Moreover, the reporting needs often evolve over time, and it is difficult

for the various developer groups to keep track of the ever-changing reporting requirements and formats and to rewrite their reporting software in response to the changes.

5   **SUMMARY OF THE INVENTION**

In view of the foregoing, the present invention provides a uniform, easily extensible, performance reporting framework that includes a plurality of reporting clients that concentrate on tracking and reporting performance data for

10   various system components and one or more reporting servers for receiving the collected data from the reporting clients and generating performance reports from the received data. Each reporting client tracks component-specific metrics of interest for monitoring one or more system components. In

15   addition, each of the reporting clients preferably tracks a core set of system attributes, such as memory usage and event logs, to facilitate system health analysis. Extensibility of the data collection is provided by the use of a plug-in for each reporting client. The reporting plug-in allows a user to

20   define and register component-specific metrics and include code for tracking and providing the current status of the metrics on demand. Moreover, the plug-in module may also be used to specify how the data on the metrics are to be stored and presented in the performance report generated by the

25   reporting server. The reporting server is configurable to provide summary reports from data reported by the reporting clients as well as more detailed reports on a per-component

basis and multi-component basis. The use of the reporting
servers to handle the automated generation of performance
reports provides uniformity in reporting formats and makes it
easy to accommodate changing reporting requirements without
5   the need to modify the underlying reporting clients.
Reporting servers may forward performance data to a reporting
super-server on a higher level of the reporting framework.
The super-server may apply a higher level of abstraction to
the data reported by the reporting servers to generate a low-
10   detail high-level summary appropriate for reviewing the health
and/or status of multiple components. The performance data
may be selectively archived in a database to facilitate
evaluation and trend analysis.

        Additional features and advantages of the invention will
15   be made apparent from the following detailed description of
illustrative embodiments, which proceeds with reference to the
accompanying figures.

                    BRIEF DESCRIPTION OF THE DRAWINGS

20   While the appended claims set forth the features of the
present invention with particularity, the invention, together
with its objects and advantages, may be best understood from
the following detailed description taken in conjunction with
the accompanying drawings of which:

25   Figure 1 is a block diagram generally illustrating an
exemplary computer on which the present invention may be
implemented;

FIG. 2 is a schematic diagram showing a network with a plurality of performance monitoring hosts;

FIG. 3 is a schematic diagram showing an embodiment of a reporting network in accordance with the invention that includes multiple reporting clients and reporting servers;

FIG. 4 is a schematic diagram showing a reporting client and a reporting server in the context of a system component stress study;

FIG. 5 is a schematic diagram showing a user-interface dialog box for a user to set up a reporting server;

FIG. 6 is a schematic diagram showing a user-interface dialog box for a user to set up a reporting client;

FIG. 7 is an exemplary summary report generated by a reporting server;

FIG. 8 is an exemplary per-component detailed report generated by the reporting server; and

FIG. 9 is a flowchart showing a process of implementing a plug-in module for a reporting client in an embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

Turning to the drawings, wherein like reference numerals refer to like elements, the invention is illustrated as being implemented in a suitable computing environment. Although not required, the invention will be described in the general context of computer-executable instructions, such as program modules, being executed by a personal computer. Generally,

program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. Moreover, those skilled in the art will appreciate that the invention

5   may be practiced with other computer system configurations, including hand-held devices, multi-processor systems, microprocessor based or programmable consumer electronics, network PCs, minicomputers, mainframe computers, and the like.

The invention may also be practiced in distributed computing

10   environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

With reference to Fig. 1, an exemplary system for

15   implementing the invention includes a general purpose computing device in the form of a conventional personal computer 20, including a processing unit 21, a system memory 22, and a system bus 23 that couples various system components including the system memory to the processing unit 21. The

20   system bus 23 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory includes read only memory (ROM) 24 and random access memory (RAM) 25. A basic input/output system

25   (BIOS) 26, containing the basic routines that help to transfer information between elements within the personal computer 20, such as during start-up, is stored in ROM 24. The personal

computer 20 further may include a hard disk drive 27 for reading from and writing to a hard disk 60, a magnetic disk drive 28 for reading from or writing to a removable magnetic disk 29, and an optical disk drive 30 for reading from or

5   writing to a removable optical disk 31 such as a CD ROM or other optical media.

The hard disk drive 27, magnetic disk drive 28, and optical disk drive 30 are connected to the system bus 23 by a hard disk drive interface 32, a magnetic disk drive interface

10   33, and an optical disk drive interface 34, respectively. The drives and their associated computer-readable media provide nonvolatile storage of computer readable instructions, data structures, program modules and other data for the personal computer 20. Although the exemplary environment described

15   herein employs a hard disk 60, a removable magnetic disk 29, and a removable optical disk 31, it will be appreciated by those skilled in the art that other types of computer readable media which can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video

20   disks, Bernoulli cartridges, random access memories, read only memories, and the like may also be used in the exemplary operating environment.

A number of program modules may be stored on the hard disk 60, magnetic disk 29, optical disk 31, ROM 24 or RAM 25,

25   including an operating system 35, one or more applications programs 36, other program modules 37, and program data 38. A user may enter commands and information into the personal

computer 20 through input devices such as a keyboard 40 and a pointing device 42. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often

5 connected to the processing unit 21 through a serial port interface 46 that is coupled to the system bus, but may be connected by other interfaces, such as a parallel port, game port or a universal serial bus (USB). A monitor 47 or other type of display device is also connected to the system bus 23

10 via an interface, such as a video adapter 48. In addition to the monitor, personal computers typically include other peripheral output devices, not shown, such as speakers and printers.

The personal computer 20 may operate in a networked

15 environment using logical connections to one or more remote computers, such as a remote computer 49. The remote computer 49 may be another personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above

20 relative to the personal computer 20, although only a memory storage device 50 has been illustrated in Fig. 1. The logical connections depicted in Fig. 1 include a local area network (LAN) 51 and a wide area network (WAN) 52. Such networking environments are commonplace in offices, enterprise-wide

25 computer networks, intranets and the Internet.

When used in a LAN networking environment, the personal computer 20 is connected to the local network 51 through a

network interface or adapter 53. When used in a WAN networking environment, the person computer 20 typically includes a modem 54 or other means for establishing communications over the WAN 52. The modem 54, which may be

5   internal or external, is connected to the system bus 23 via the serial port interface 46. In a networked environment, program modules depicted relative to the personal computer 20, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network

10 connections shown are exemplary and other means of establishing a communications link between the computers may be used.

In the description that follows, the invention will be described with reference to acts and symbolic representations

15 of operations that are performed by one or more computers, unless indicated otherwise. As such, it will be understood that such acts and operations, which are at times referred to as being computer-executed, include the manipulation by the processing unit of the computer of electrical signals

20 representing data in a structured form. This manipulation transforms the data or maintains it at locations in the memory system of the computer, which reconfigures or otherwise alters the operation of the computer in a manner well understood by those skilled in the art. The data structures where data is

25 maintained are physical locations of the memory that have particular properties defined by the format of the data. However, while the invention is being described in the

foregoing context, it is not meant to be limiting as those of skill in the art will appreciate that various of the acts and operation described hereinafter may also be implemented in hardware.

5   Referring now to FIG. 2, the present invention is directed to an improved way to collect and report system component performance data that provides extensibility in data collection and consistency, ease, and uniformity in reporting the results of the performance monitoring. As shown in FIG.

10   2, a networked computer system 70 may include many performance-monitoring machines 71-76, each of which may be directed to performance reporting for a specific component of the system. A component may be considered as a binary image or a set of binary images that work together to provide a

15   service. Examples of such services in a networking context include a telnet server, a domain name system (DNS) server, services for MacIntosh routing, remote access server (RAS), etc. Examples of other types of services include audio and video recording/playback, USB device support, windowing

20   services, file system management, and memory management. Because the monitoring of a given component of the system may require collection of statistical data specific to the component, it is highly desirable to enable a user responsible for the given component to define, collect, and report such

25   component-specific data. On the other hand, in order to effectively use the collected data to assist performance review and system diagnosis, the data are to be reported in

useful formats that summarize the data in suitable levels of abstraction. Moreover, it is desirable to have the performance data collected for different system components to be reported in a standardized manner. In this regard, it is

5   also highly desirable to minimize the burden on the users responsible for the system components in reporting the collected data and generating the performance reports.

The present invention effectively addresses these needs in reporting the performance of components of a computer

10   system by providing a framework for automated performance reporting. As shown in FIG. 3, the reporting system in accordance with the invention includes multiple layers directed to different levels of abstraction for performance reporting. The bottom layer 82 includes a plurality of

15   reporting clients 83-89 that are responsible for collecting statistical data relating to network performance of different system components. The next layer 90 of the reporting system includes, depending on the complexity of the network, one or more reporting servers, such as servers 92-94. Each reporting

20   server is responsible for receiving the statistical data collected by one or more reporting clients and presenting the received data in a report. For instance, in the illustrated example of FIG. 3, the reporting servers 92 receives performance data from reporting clients 83, 84 and generates a

25   report 100. The report 100 may include a summary 102 that summarizes the performance data in a level of abstraction suitable for the managerial review intended for the report,

and may also include a more detailed report 106 for each of the reporting clients reporting to the server. The reporting system may optionally have higher levels of reporting servers that receive data from reporting servers on a lower layer and

5   generating a report of a higher level of abstraction than those of the lower level servers. For example, the embodiment of FIG. 3 includes a server 112 functioning as a "super-server" that receives data reported by the servers 92, 93, and 94 on the lower layer 90. The super-server 112 preferably can

10 perform all the reporting functions of the reporting servers, plus the ability to generate a low-detail high-level report 116 suitable for reviewing the health or status of multiple sets of system components. It will be appreciated that more super-servers or servers on even higher layers may be provided

15 in the framework if needed to accommodate the reporting needs of the network being monitored. Moreover, a reporting server does not have to be restricted to a single layer of the hierarchy. For example, the server 112 may function both as a reporting super-server in relation to the servers 92, 93, 94,

20 and also as a server that receives reports directly from reporting clients, such as the reporting client 85.

In accordance with an aspect of the invention, each of the reporting servers on the different layers may store performance report data and other data in a data store 120,

25 such as an SQL database. Archiving the performance reporting data, which may be raw data or summarized data, allows easy

evaluation and trending of data gathered across multiple
reporting clients on a given set of hosts in the network.

The automated network performance report generation
provided by the reporting framework according to the invention
5   significantly simplifies the task of coalescing the collected
statistical data into useful reports. By using report servers
to handle report generation, the users of the reporting
clients are allowed to concentrate on the collection and
analysis of statistical data specific to the system component
10   they are testing or studying. The use of report servers to
generate reports also allows reports for various system
components to be generated in a uniform manner with
standardized formats. The reporting of component-specific
data and attributes is supported, with the data presented in
15   standardized, well-defined, places in the report that can be
easily read and understood. Moreover, by using multiple
layers of reporting servers, reports that summarize the system
component performance in different levels of abstraction can
be automatically generated to suit the information needs of
20   various levels of management.

The division of the reporting system into reporting
clients for collecting data and reporting servers for
generating reports also makes it easier to modify the
reporting system to accommodate changing reporting
25   requirements. If the formats of the reports are to be
changed, only the reporting servers have to be modified to
implement the changes. To that end, a modified reporting

server application that automatically generates reports in the new format may be redistributed for installation on the machines running the reporting servers. The reporting clients, on the hand, do not have to be affected if the

5   changes to the reports do not require modifications to their collection of statistical data.

In accordance with a feature of the invention, each reporting client is provided with the flexibility of defining the types of statistical data it will collect and report to an

10   associated reporting server. To that. end, each reporting client preferably includes a client application or module and a plug-in module. The client application provides baseline monitoring functionality and may be configured to selectively collect a set of "baseline" system attributes data, such as

15   event logs, per-process memory usage and CPU time, etc. The plug-in, on the other hand, defines the specific performance data to be collected for a given system component and handles the collection of such component-specific data. In a preferred embodiment, these component-specific data are

20   registered with the client module of the reporting client, which in turn registers the component-specific data with its associated reporting server. By providing core monitoring functionality in the reporting client, the reporting system allows critical system attributes to be uniformly monitored

25   and rolled-up.

The component-specific data collected and reported by a reporting client can vary significantly depending on the

nature of the system component monitored by the client.

Generally, the data may include quantifiable statistical data

commonly referred to as "metrics" and other data commonly

called "free-form" data that are not readily amenable to

5   quantification. For example, user-defined metrics may include

number of connections, service response time, bytes of storage

remaining, number of database queries, etc. The free-form

data may include, for example, a list of network clients that

have connected to a host in the last 24 hours, progress on

10   important calculations, etc.

For illustration purposes, the following description

describes a preferred embodiment implemented in the context of

software development of an operating system with networking

functionality. Although this embodiment shows an example of

15   network performance monitoring in the context of software

development, it will be appreciated that the invention may

also be used in monitoring the performance of an existing

network. In this embodiment, a plurality of networked

computers (or "hosts") running the operating system are

20   subjected to stress testing. Each computer is directed to

monitoring how one or more system components of the operating

system perform in the stress tests, and the operation data

relevant for the performance of that component are collected

and reported for analysis and review. In this regard,

25   referring to FIG. 4, a host 128 involved in the stress study

may have one or more reporting clients 132 each operating as a

"stress server." Each stress server tracks stress test data

that include a core set of system attributes as well as information specific to the system component monitored by the stress server. The core set of system attributes includes, for instance, host configuration, build number, memory and

5   handle usage, and event logs generated during stress. The stress data 134 collected by the stress server 132 is then reported to its associated reporting server 136 through a named pipe 138.

In accordance with an aspect of the embodiment, to

10   minimize the efforts required in implementing the reporting system, the software application or module 140 implementing the reporting server is prepared and distributed to the groups of the development team responsible for different networking components. One computer of each group is designated as the

15   reporting server 136 and runs the reporting server application 140. Likewise, a standard client application or module 144 is prepared and distributed for running on each host computer 128 in the stress study. The client application 144 is configurable by a user 146 responsible for testing the given

20   system component for selectively tracking the system attributes. The user is only required to provide a plug-in module 150 for use with the provided client application 144 to track and report component-specific stress data. In this regard, to help the user 146 in developing the component-

25   specific plug-in 150, preferably a sample plug-in module that can be easily modified to define and track the component-specific stress data is distributed to different component

groups. The user can then modify the sample reporting server
to provide tracking of the component-specific metrics, as will
be described in greater detail below.

After the server application 140 is loaded onto a
5   designated reporting server host 152, it is set up for the
reporting task in the specific environment. To simplify the
task of setting up the reporting server, user-interface
including set-up dialog boxes is preferably provided to guide
a user in providing the needed set-up information. In one
10 implementation, a user is presented with a set-up dialog box
by selecting a "View-> Server Settings" menu item. As showing
in the exemplary dialog box 160 of FIG. 5, the dialog box
includes a plurality of check-boxes for and fields for the
user to configure the reporting server.

15   In this illustrated example, the first portion 168 of the
dialog box 160 is for the user to select the information to be
included in header portion the report generated by this report
server. The Group Name field 170 is for the user to enter the
name of the software development group that will use this
20 report server for performance reporting. The checkboxes 161-
166 are for the user to select information describing the
machine on which the reporting server resides for inclusion in
the stress report. When the "Machine Info." box 161 is
checked, a server details header in the report will include a
25 "MACHINE:" line that includes the server name, processor
architecture, number of processors, build, and type of build.
The server details hearer is used to provide additional system

configuration information on a per-server basis, such as the

software version, machine name, and debugger used for that

machine, etc. Checking the "Debugger Remote Line" box 162

will cause the report header to include the remote server

5   information for connecting to that server's debugger. When

the "Machine Up Time" box 163 is checked, the report header

will include an "UPTIME" line in the header. When the

"Status" box 164 is checked, a "STATUS:" line will be included

in the header. This line indicates whether the stress test is

10   considered as PASSED, FAILED, LOST, or any other status listed

in a ProvideReportBuffer function as will be described in

greater detail below. When the "Summary" box 165 is checked,

an overall stress summary table showing the name and status of

all servers in the stress testing is printed at the top of the

15   stress report. When the "Long Haul Report" box 166 is

checked, any long haul servers involved in the stress testing

are summarized at the top of the report. An example of a

machine header is provided as follows

```
20  01.   MACHINE   :   JOHMIL-DTC1, x86 UP, 2116 FREE
          REMOTE    :   REMOTE/c johnmil-lap JOHNMIL-DTC1
          UPTIME    :   3:45;25
          PRIVATES  :   fish.sys
          SUMDATA   :   //JOHNML-DTC1:      2116+P 3:45:25
25
```

As to the other fields in the dialog box 160, the

Reporting Server Pipe Name field 172 allows the user to

specify a named pipe for a reporting client to communicate

with the reporting server. The Report File Path field 174 is

30   for specifying the directory where the reporting server is to

save the stress report it generates. After the server

receives data from a reporting client, it outputs all received

data to this file. The Report File Extension field 176 allows

the user to enter a file name extension for distinguishing the

5   stress reports for the components of the user's group from

those for other components. The Summary FILE is usually a

single strictly formatted line, such as:

SFM Servers   1   .   1   (100%)

10

In the illustrated embodiment, there are two basic

designations of stress servers: nightly stress servers that

run overnight and are then reset, and long-haul stress servers

that are run until they fail or until the build being tested

15   is so old that testing it is no longer useful. These two

types of servers are tallied differently because the failures

encountered indicate different types of issues. The number of

servers that are being monitored (i.e., the number of

reporting clients) does not include long-haul servers in this

20   embodiment. The Summary File Description field 182 is for

entering the name of the component to be monitored. The Super

Server Computer Name 184 field allows the user to specify a

super server that creates stress summaries from multiple

component stress reports. The Super Server Pipe field 186 is

25   for specifying a named pipe for the reporting server to

communicate with the super server.

After the reporting server 152 is set up, a reporting
client application 144 is set up on each of the stress
servers. User Interface is also preferably used for allowing
the user to enter setup information for the reporting client.

5   In one implementation, a Client Setting dialog box is shown by
choosing the View->Client Settings menu item. An exemplary
Client Setting dialog box 190 is shown in FIG. 6. The
Reporting Server Computer Name field 192 in this dialog box is
for entering the name of the computer that the reporting

10   server is running on. The reporting client uses this name as
part of the pipe name to find and communicate with the
reporting server 136. All reporting clients for the stress
testing of a given system component preferably report the data
they collect to a single reporting server. The Reporting

15   Server Pipe Name field 194 is the name of the pipe given in
the reporting server settings. The reporting client and the
reporting server should use the same named pipe to avoid
communication failure. The Debugger Name field 196 is for
entering the name of a kernel debugger attached to the

20   computer that the reporting client application is running on.
The Customer Report Buffer Name field 198 is for the user to
specify the name of a dynamic link library (DLL) file that
functions as a plug-in for the client application. As
described above, the DLL is provided for defining and tracking

25   component-specific data. The long-haul Server checkbox 200
when checked indicates that the machine running the client
application is a long-haul stress server. The reporting

server can use this information to provide a separate summary
block just for this stress server. A sample long-haul summary
block in the report by the reporting server is provided below:

```
 5  Longhaul Summary: 1 servers 1 up (100%)
    MACHINE    :    SFM_LONGHAUL_S
    BUILD      :    2059
    UPTIME     :    86:58:05
    MAX UPTIME:    436:42:25
10  MAX BUILD  :    2048
    COMPONENTS:    MacFile, MacPrint, ATCP, PPP
```

The Components Tested field 202 is where the names of the
components being stress tested under the long-haul scenario
15 are entered.

In the MISC. block 204 of the dialog box, the first field
206 is for the user to enter any "privates" being run on the
stress machine. Software development for large projects is
usually done by collecting changes over a period of time (such
20 as a day), compiling a full version of the product at that
point, then releasing the generated binaries as a 'build'.
Builds are often used by the entire team to test the product,
and, in the case of operating systems, to develop the product
as well. Developers often want to test unusually complex or
25 risky fixes before making them part of the build, since
checking bad code into the build can set the entire project
back. To do this, they compile their own copy of any relevant
binaries, and then replace the corresponding binaries in the
current build with their private copies called "privates". A
30 private may be a single binary, or a set of binaries. The
privates are listed in the machine header in the report

generated by the reporting server. The second field 208 in the MISC. block allows the user to provide a short description of the type of the stress testing occurring on the stress server machine. For example, if the user's group is operating

5 a "Services For Macintosh" stress test, the user may enter "Mac File Stress" or "AppleTalk Router." Once the user has updated the settings for the reporting client and pressed the "OK" button in the dialog box, the reporting client immediately attempts to connect to the reporting server.

10 As mentioned above, in this embodiment for stress testing, the reporting client provides consistent tracking for both a core set of system attributes and component-specific attributes. The core set of system attributes includes, for example, memory and handle usage and event logging. The

15 memory, handle, and event log tracking are most effective when they are limited to elements relevant to the component being stressed. By default, no memory and handle usage will be tracked, but all events generated during the stress testing will be tracked.

20 In addition to the core set of system attributes, the reporting client tracks statistics specific to the component being stress tested by providing the component-specific DLL on the reporting client. The quantifiable component-specific attributes, commonly referred to as "metrics," can be a

25 variety of different things defined by the user. The only requirement in this embodiment for a metric is that it be an integer numeric quantity that fits in a longlong variable.

When each reporting client application is started, it queries its component DLL 150 for information regarding the component-specific metrics it wants to track. To communicate with the reporting server for this purpose, the component DLL

5   implements specific functions that will be described in greater detail below. For example, one of the functions implemented by the DLL is for registering a label for each of the metrics (e.g., "# of Client Connections").

In accordance with a feature of the embodiment, the

10   component DLL also indicates how the data regarding the metric are to be reported. For example, referring to FIG. 4, the DLL 150 can specify whether the current value of a metric, its maximum value, or its average should be reported in the stress report. The reporting client application 144 also queries the

15   component DLL 150 to learn whether the component has passed or failed the stress testing. In this regard, the component DLL can programmatically determine the pass or fail status of the component stress testing, and returning an appropriate value to indicate the status. For example, if a web server being

20   stressed is no longer sending pages to web clients, the DLL for that component may return a value indicating that the web server has failed the stress test.

In accordance with an aspect of the invention, the reporting server 136 may provide a summary of the stress

25   testing results of the reporting client that reports to it, as well as more detailed reports for each of the reporting client. For illustration purposes, an exemplary stress

summary table 210 is provided in FIG. 7, and a detailed report for one of the stress servers listed in the stress summary table is provided in FIG. 8. The stress summary table identifies the machines on which the stress test was run,

5   their platform and build, the stress test conducted, and their respective pass/fail status. One record 212 in the summary table 210 shows that a machine named R-1C8 has passed the stress test.

The detailed per-client report 216 generated by the

10   reporting server 136 for this R-1C8 machine as shown in FIG. 8 includes a Metrics Report section 218, a Kernel Memory Usage section 220, a User Memory + Handle Usage section 222, and an Event Log Errors section 224. The Metrics Report section 218 lists the descriptive labels for the metrics tracked and their

15   respective values. The Event Log Errors section 224 identifies for each event log error the count, event ID, the component involved, and the event text.

Turning now to the preparation of a component DLL for performance reporting, each component DLL 150 exports well-

20   known functions to the reporting client application 144 for communication therewith. These functions return component specific information and can also be used to modify the behavior of the reporting server 136. As mentioned above, in a preferred implementation a sample report DLL is provided to

25   the users of the reporting clients. The sample DLL, which includes code implementing all required entry points, can be

easily modified to suit the data collecting and reporting requirements for a specific component.

Referring now to FIG. 9, in developing the component DLL, the user implements a SetComponentEvtTypes function in the DLL

5   for registering with the client application how much detail the user wants on the event logs to be tracked (step 228). A function SetComponentProcesses is implemented to register process memory tracking options and memory tracking presentation options (step 230). A function

10   SetComponentPoolTags is implemented for registering kernel memory tracking if the user has kernel memory pool tagging enabled and wants to track memory usage for specific kernel-mode components (step 232). If the component does not already have an assigned ordinal ID, the user makes an ordinal ID for

15   her component and implements the function RegisterComponentID for registering that component ID (step 234).

The user also determines the component-specific metrics to be tracked and reported (step 236). A function RegisterUnitsCount is implemented for registering the number

20   of metrics the user wants to track, and another function RegisterUnit is implemented for registering the labels for the component-specific metrics and the way each of the metrics should be summarized by the reporting server (step 238). To track the metrics, the user includes code in the DLL for

25   deriving the current value of each of the metrics (step 240).

If the user has already implemented in an old stress-reporting tool the code for tracking the metrics, such old

code may be reused by simply cutting and pasting it into the component DLL. The reporting of the metrics values is achieved by implementing a function SetUnitValue (step 242).

If the pass/fail status of the stress testing of the

5 component is to be programmatically determined, the user modifies a ProvideReportBuffer function that performs the determination and reports the pass/fail result (step 244). This function can also be used to pass any out-of-band data that are not readily quantifiable and therefore not amenable

10 to being treated as metrics.

The well-known functions defined in this embodiment for use in the component-specific DLL, including those functions mentioned above in describing the DLL development process, are summarized below. The required functions in this embodiment

15 and their purposes are as follows:

GetConfigInfo: For providing version information

ProvideReportBuffer: For providing text data to be included

in the per-client report and the pass/fail status of the

20 component.

The optional functions and their purposes are as follows:

SetComponentEvtTypes: For limiting types of event logs to be

25 tracked by the client application and specifying the

degree to which the events encountered are displayed in

the report.

SetComponentProcesses: For specifying the process names for

    memory/handle tracking and specifying the line items to

    be displayed for each process.

SetComponentPoolTags: For specifying kernel pool tags to

5   track memory for tracking if pool tagging is enabled.

RegisterComponentID: For identifying the component this

    reporting DLL is written to track.

RegisterUnitsCount: For providing the number of component-

    specific metrics/units to be tracked.

10 RegisterUnit: For enumerating labels for the component-

    specific metrics and the way each metric should be

    summarized and/or presented.

SetUnitValue : For reporting the current value of a

    component-specific metric.

15 Config: For invoking a dialog to set any component-specific

    configuration.

    In view of the many possible embodiments to which the

principles of this invention may be applied, it should be

20 recognized that the embodiment described herein with respect

to the drawing figures is meant to be illustrative only and

should not be taken as limiting the scope of invention. For

example, those of skill in the art will recognize that the

elements of the illustrated embodiment shown in software may

25 be implemented in hardware and vice versa or that the

illustrated embodiment can be modified in arrangement and

detail without departing from the spirit of the invention.

Therefore, the invention as described herein contemplates all such embodiments as may come within the scope of the following claims and equivalents thereof.